

## HW5 : factorial 함수 구현

20160394 임효상

### 1) 실습 A

Debugger window showing assembly code and register values. The assembly code is for a factorial function. The register window shows r0 = 0x2, sp = 0xffff128, and other registers.

Register	Value	Comment
r0	0x2	2
r1	0x0	0
r2	0x0	0
r3	0x0	0
r4	0x10d14	68884
r5	0x10158	65880
r6	0x8b408	570376
r7	0x10158	65880
r8	0x0	0
r9	0x0	0
r10	0x89b04	563972
r11	0xffff154	4294898004
r12	0x25	37
sp	0xffff128	4294897960
lr	0x1054c	66892

처음 r0 = 2, sp = 0xffff128 이다.

Debugger window showing assembly code and register values after the first instruction. The register window shows sp = 0xffff120, indicating a stack pointer adjustment.

Register	Value	Comment
r2	0x0	0
r3	0x0	0
r4	0x10d14	68884
r5	0x10158	65880
r6	0x8b408	570376
r7	0x10158	65880
r8	0x0	0
r9	0x0	0
r10	0x89b04	563972
r11	0xffff154	4294898004
r12	0x25	37
sp	0xffff120	4294897952
lr	0x1054c	66892

sub sp, sp, #8 명령어 수행 -> SUB명령어로 sp 값이 8만큼 감소되었다.

factRecursive:				Tree
sub sp, sp, #8				memory
str lr, [sp, #4]				0xfffff110 0xfffff14f 4
str r0, [sp, #0]				address hex char
cmp r0, #1				more
bge loop				0xfffff110 58 01 01 00 X...
mov r0, #1				0xfffff114 00 00 00 00 ....
add sp, sp, #8				0xfffff118 00 00 00 00 ....
mov pc,lr				0xfffff11c 04 9b 08 00 ....
loop:				0xfffff120 54 f1 fe ff T...
sub r0, r0, #1	0x10544 subtr0, r0,			0xfffff124 4c 05 01 00 L...
bl factRecursive	0x10548 blt0x10524			0xfffff128 03 00 00 00 ....
mov r12, r0	0x1054c movtr12, r0			0xfffff12c 4c 05 01 00 L...
ldr r0, [sp, #0]	0x10550 ldrr0, [sp,			0xfffff130 04 00 00 00 ....
ldr lr, [sp, #4]	0x10554 ldrtlr, [sp,			0xfffff134 4c 05 01 00 L...
add sp, sp, #8	0x10558 addtsp, sp,			0xfffff138 05 00 00 00 ....
mul r0, r12, r0	0x1055c multtr0, r12,			0xfffff13c 4c 05 01 00 L...
mov pc, lr	0x10560 movtpc, lr			
.end				
(end of file)				

str lr, [sp, #4] 명령어 수행 -> 현재 sp값 + 4의 위치에 link register 값을 저장했다. link register의 값은 이전에 0x1054c로 저장되었다.

1 .text				expressions
2 .global factRecursive				Tree
3				memory
4 factRecursive:				0xfffff110 0xfffff14f 4
5 sub sp, sp, #8				address hex char
6 str lr, [sp, #4]				more
7 str r0, [sp, #0]				0xfffff110 58 01 01 00 X...
8 cmp r0, #1				0xfffff114 00 00 00 00 ....
9 bge loop				0xfffff118 00 00 00 00 ....
10 mov r0, #1				0xfffff11c 04 9b 08 00 ....
11 add sp, sp, #8				0xfffff120 02 00 00 00 ....
12 mov pc,lr				0xfffff124 4c 05 01 00 L...
13 loop:				0xfffff128 03 00 00 00 ....
14 sub r0, r0, #1				0xfffff12c 4c 05 01 00 L...
15 bl factRecursive				0xfffff130 04 00 00 00 ....
16 mov r12, r0				0xfffff134 4c 05 01 00 L...
17 ldr r0, [sp, #0]				0xfffff138 05 00 00 00 ....
18 ldr lr, [sp, #4]				0xfffff13c 4c 05 01 00 L...
19 add sp, sp, #8				0xfffff140 06 00 00 00 ....
20 mul r0, r12, r0				
21 mov pc, lr				
22 .end				
23				
24 (end of file)				

str lr, [sp, #0] 명령어 수행 -> 현재 sp값 위치에 r0(2)를 저장했다.

factRecursive:			
sub	sp, sp, #8		
str	lr, [sp, #4]		
str	r0, [sp, #0]		
cmp	r0, #1		
bge	loop		
mov	r0, #1		
add	sp, sp, #8		
mov	pc, lr		
loop:			
sub	r0, r0, #1	0x10544 subtr0, r0,	
bl	factRecursive	0x10548 blt0x10524	
mov	r12, r0	0x1054c movtr12, r0	
ldr	r0, [sp, #0]	0x10550 ldtr0, [sp,	
ldr	lr, [sp, #4]	0x10554 ldrtlr, [sp,	
add	sp, sp, #8	0x10558 addtsp, sp,	
mul	r0, r12, r0	0x1055c multtr0, r12,	
mov	pc, lr	0x10560 movtpc, lr	
.end			
(end of file)			

  

name	value (hex)	(decimal)	description
r0	0x2	2	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	register 8 (64-bit)
r9	0x0	0	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)
r11	0xfffff154	4294898004	register 11 (64-bit)
r12	0x25	37	register 12

cmp r0, #1과 bge loop 명령어 수행 -> r0와 1의 값을 비교, r0값이 크므로 loop로 branch한다.

1	.text			send	SIGINT	to	gdb (pid 40662)	debug program (pid 1)
2	.global factRecursive			other pid	pid			
3								
4	factRecursive:							
5	sub	sp, sp, #8						
6	str	lr, [sp, #4]						
7	str	r0, [sp, #0]						
8	cmp	r0, #1						
9	bge	loop						
10	mov	r0, #1						
11	add	sp, sp, #8						
12	mov	pc, lr						
13	loop:							
14	sub	r0, r0, #1	0x10544 subtr0, r0,					
15	bl	factRecursive	0x10548 blt0x10524					
16	mov	r12, r0	0x1054c movtr12, r0					
17	ldr	r0, [sp, #0]	0x10550 ldtr0, [sp,					
18	ldr	lr, [sp, #4]	0x10554 ldrtlr, [sp,					
19	add	sp, sp, #8	0x10558 addtsp, sp,					
20	mul	r0, r12, r0	0x1055c multtr0, r12,					
21	mov	pc, lr	0x10560 movtpc, lr					
22	.end							
23								
24	(end of file)							

  

name	value (hex)	value (decimal)	description
r0	0x1	1	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	register 8 (64-bit)
r9	0x0	0	register 9 (64-bit)
r10	0x89b04	563972	register 10 (64-bit)

sub r0,r0,#1 명령어 수행 -> r0의 값을 1만큼 감소시켰다. 이제 r0는 1이다.

<pre> 1 .text 2 .global factRecursive 3 4 factRecursive: 5     sub    sp, sp, #8 6     str    lr, [sp, #4] 7     str    r0, [sp, #0] 8     cmp    r0, #1 9     bge    loop 10    mov    r0, #1 11    add    sp, sp, #8 12    mov    pc,lr 13 loop: 14    sub    r0, r0, #1           0x10544 subtr0, r0, 15    bl     factRecursive       0x10548 blt0x10524 &lt; 16    mov    r12, r0             0x1054c movtr12, r0 17    ldr    r0, [sp, #0]        0x10550 ldrtr0, [sp] 18    ldr    lr, [sp, #4]        0x10554 ldrtlr, [sp] 19    add    sp, sp, #8          0x10558 addtsp, sp, 20    mul    r0, r12, r0         0x1055c multr0, r12, 21    mov    pc, lr             0x10560 movtpc, lr 22 .end 23 24 (end of file) </pre>		<div>Breakpoints</div> <div>signals</div> <div> <div>send</div> <div>SIGINT ▾</div> <div>to</div> <div>gdb (pid 40662)</div> <div>debug program (pid 1)</div> </div> <div> <div>other pid</div> <div>pid</div> </div>																																																	
		<div>registers</div> <table> <tr> <th>name</th><th>value (hex)</th><th>value (decimal)</th><th>description</th></tr> <tr><td>r0</td><td>0x1</td><td>1</td><td></td></tr> <tr><td>r1</td><td>0x0</td><td>0</td><td></td></tr> <tr><td>r2</td><td>0x0</td><td>0</td><td></td></tr> <tr><td>r3</td><td>0x0</td><td>0</td><td></td></tr> <tr><td>r4</td><td>0x10d14</td><td>68884</td><td></td></tr> <tr><td>r5</td><td>0x10158</td><td>65880</td><td></td></tr> <tr><td>r6</td><td>0x8b408</td><td>570376</td><td></td></tr> <tr><td>r7</td><td>0x10158</td><td>65880</td><td></td></tr> <tr><td>r8</td><td>0x0</td><td>0</td><td>register 8 (64-bit)</td></tr> <tr><td>r9</td><td>0x0</td><td>0</td><td>register 9 (64-bit)</td></tr> <tr><td>r10</td><td>0x89b04</td><td>563972</td><td>register 10</td></tr> </table>		name	value (hex)	value (decimal)	description	r0	0x1	1		r1	0x0	0		r2	0x0	0		r3	0x0	0		r4	0x10d14	68884		r5	0x10158	65880		r6	0x8b408	570376		r7	0x10158	65880		r8	0x0	0	register 8 (64-bit)	r9	0x0	0	register 9 (64-bit)	r10	0x89b04	563972	register 10
name	value (hex)	value (decimal)	description																																																
r0	0x1	1																																																	
r1	0x0	0																																																	
r2	0x0	0																																																	
r3	0x0	0																																																	
r4	0x10d14	68884																																																	
r5	0x10158	65880																																																	
r6	0x8b408	570376																																																	
r7	0x10158	65880																																																	
r8	0x0	0	register 8 (64-bit)																																																
r9	0x0	0	register 9 (64-bit)																																																
r10	0x89b04	563972	register 10																																																

**bl factRecursive** 명령어 수행 -> branch 명령에 따라 factRecursive로 branch한다. link register에 다음 명령(mov r12,r0)의 pc값 0x1054c를 저장한다.

<pre> 1 .text 2 .global factRecursive 3 4 factRecursive: 5     sub    sp, sp, #8 6     str    lr, [sp, #4] 7     str    r0, [sp, #0] 8     cmp    r0, #1 9     bge    loop 10    mov    r0, #1 11    add    sp, sp, #8 12    mov    pc,lr 13 loop: 14    sub    r0, r0, #1           0x10544 subtr0, r0, 15    bl     factRecursive       0x10548 blt0x10524 &lt; 16    mov    r12, r0             0x1054c movtr12, r0 17    ldr    r0, [sp, #0]        0x10550 ldrtr0, [sp] 18    ldr    lr, [sp, #4]        0x10554 ldrtlr, [sp] 19    add    sp, sp, #8          0x10558 addtsp, sp, 20    mul    r0, r12, r0         0x1055c multr0, r12, 21    mov    pc, lr             0x10560 movtpc, lr 22 .end 23 24 (end of file) </pre>		<table> <tr><td>r0</td><td>0x1</td><td>1</td><td></td></tr> <tr><td>r1</td><td>0x0</td><td>0</td><td></td></tr> <tr><td>r2</td><td>0x0</td><td>0</td><td></td></tr> <tr><td>r3</td><td>0x0</td><td>0</td><td></td></tr> <tr><td>r4</td><td>0x10d14</td><td>68884</td><td></td></tr> <tr><td>r5</td><td>0x10158</td><td>65880</td><td></td></tr> <tr><td>r6</td><td>0x8b408</td><td>570376</td><td></td></tr> <tr><td>r7</td><td>0x10158</td><td>65880</td><td></td></tr> <tr><td>r8</td><td>0x0</td><td>0</td><td>register 8 (64-bit)</td></tr> <tr><td>r9</td><td>0x0</td><td>0</td><td>register 9 (64-bit)</td></tr> <tr><td>r10</td><td>0x89b04</td><td>563972</td><td>register 10 (64-bit)</td></tr> <tr><td>r11</td><td>0xfffff154</td><td>4294898004</td><td>register 11 (64-bit)</td></tr> <tr><td>r12</td><td>0x25</td><td>37</td><td>register 12 (64-bit)</td></tr> <tr><td>sp</td><td>0xfffff118</td><td>4294897944</td><td></td></tr> <tr><td>lr</td><td>0x1054c</td><td>66892</td><td></td></tr> </table>		r0	0x1	1		r1	0x0	0		r2	0x0	0		r3	0x0	0		r4	0x10d14	68884		r5	0x10158	65880		r6	0x8b408	570376		r7	0x10158	65880		r8	0x0	0	register 8 (64-bit)	r9	0x0	0	register 9 (64-bit)	r10	0x89b04	563972	register 10 (64-bit)	r11	0xfffff154	4294898004	register 11 (64-bit)	r12	0x25	37	register 12 (64-bit)	sp	0xfffff118	4294897944		lr	0x1054c	66892	
r0	0x1	1																																																													
r1	0x0	0																																																													
r2	0x0	0																																																													
r3	0x0	0																																																													
r4	0x10d14	68884																																																													
r5	0x10158	65880																																																													
r6	0x8b408	570376																																																													
r7	0x10158	65880																																																													
r8	0x0	0	register 8 (64-bit)																																																												
r9	0x0	0	register 9 (64-bit)																																																												
r10	0x89b04	563972	register 10 (64-bit)																																																												
r11	0xfffff154	4294898004	register 11 (64-bit)																																																												
r12	0x25	37	register 12 (64-bit)																																																												
sp	0xfffff118	4294897944																																																													
lr	0x1054c	66892																																																													

**sub sp,sp,#8** 명령어 수행 -> 기존 sp값 0xfffff120에서 8만큼 감소시켰다.



```

1 .text
2 .global factRecursive
3
4 factRecursive:
5     sub    sp, sp, #8
6     str    lr, [sp, #4]
7     str    r0, [sp, #0]
8     cmp    r0, #1
9     bge    loop
10    mov     r0, #1
11    add     sp, sp, #8
12    mov     pc,lr
13 loop:
14    sub     r0, r0, #1
15    bl      factRecursive
16    mov     r12, r0
17    ldr     r0, [sp, #0]
18    ldr     lr, [sp, #4]
19    add     sp, sp, #8
20    mul     r0, r12, r0
21    mov     pc, lr
22 .end
23
24 (end of file)

```

address	hex	char
0xfffff110	58 01 01 00	X...
0xfffff114	00 00 00 00	....
0xfffff118	01 00 00 00	....
0xfffff11c	4c 05 01 00	L...
0xfffff120	02 00 00 00	....
0xfffff124	4c 05 01 00	L...
0xfffff128	03 00 00 00	....
0xfffff12c	4c 05 01 00	L...
0xfffff130	04 00 00 00	....
0xfffff134	4c 05 01 00	L...
0xfffff138	05 00 00 00	....
0xfffff13c	4c 05 01 00	L...
0xfffff140	06 00 00 00	....
0xfffff144	b0 05 01 00	....
0xfffff148	06 00 00 00	....

cmp r0, #1과 bge loop 명령어 수행 -> r0와 1의 값을 비교, r0값이 같으므로 loop로 branch한다.

```

1 .text
2 .global factRecursive
3
4 factRecursive:
5     sub    sp, sp, #8
6     str    lr, [sp, #4]
7     str    r0, [sp, #0]
8     cmp    r0, #1
9     bge    loop
10    mov     r0, #1
11    add     sp, sp, #8
12    mov     pc,lr
13 loop:
14    sub     r0, r0, #1
15    bl      factRecursive
16    mov     r12, r0
17    ldr     r0, [sp, #0]
18    ldr     lr, [sp, #4]
19    add     sp, sp, #8
20    mul     r0, r12, r0
21    mov     pc, lr
22 .end
23
24 (end of file)

```

name	value (hex)	value (decimal)	descri
r0	0x0	0	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	registe (64-bit
r9	0x0	0	registe (64-bit

sub r0,r0,#1 명령어 수행 -> r0의 값을 1만큼 감소시켰다. 이제 r0는 0이다.

```

1  .text
2  .global factRecursive
3
4  factRecursive:
5      sub    sp, sp, #8
6      str    lr, [sp, #4]
7      str    r0, [sp, #0]
8      cmp    r0, #1
9      bge    loop
10     mov    r0, #1
11     add    sp, sp, #8
12     mov    pc,lr
13 loop:
14     sub    r0, r0, #1
15     bl     factRecursive
16     mov    r12, r0
17     ldr    r0, [sp, #0]
18     ldr    lr, [sp, #4]
19     add    sp, sp, #8
20     mul    r0, r12, r0
21     mov    pc, lr
22 .end
23
24 (end of file)

```

more
breakpoints

signals
registers

name	value (hex)	value (decimal)	descri
r0	0x0	0	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	registe (64-bit)
r9	0x0	0	registe (64-bit)

**bl factRecursive** 명령어 수행 -> branch 명령에 따라 factRecursive로 branch한다. link register에 다음 명령(mov r12,r0)의 pc값 0x1054c를 저장한다.

```

1  .text
2  .global factRecursive
3
4  factRecursive:
5      sub    sp, sp, #8
6      str    lr, [sp, #4]
7      str    r0, [sp, #0]
8      cmp    r0, #1
9      bge    loop
10     mov    r0, #1
11     add    sp, sp, #8
12     mov    pc,lr
13 loop:
14     sub    r0, r0, #1
15     bl     factRecursive
16     mov    r12, r0
17     ldr    r0, [sp, #0]
18     ldr    lr, [sp, #4]
19     add    sp, sp, #8
20     mul    r0, r12, r0
21     mov    pc, lr
22 .end
23
24 (end of file)

```

r7	0x10158	65880	
r8	0x0	0	registe (64-bit)
r9	0x0	0	registe (64-bit)
r10	0x89b04	563972	registe (64-bit)
r11	0xfffff154	4294898004	registe (64-bit)
r12	0x25	37	registe (64-bit)
sp	0xfffff110	4294897936	
lr	0x1054c	66892	
pc	0x10528	66856	
cpsr			
d0			VFP double precisi (Temp registe)
d1			VFP

**sub sp,sp,#8** 명령어 수행 -> 기존 sp값 0xfffff118에서 8만큼 감소시켰다.

	address	hex	char
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			

  

more	address	hex	char
	0xfffff110	58 01 01 00	X...
	0xfffff114	4c 05 01 00	L...
	0xfffff118	01 00 00 00	....
	0xfffff11c	4c 05 01 00	L...
	0xfffff120	02 00 00 00	....
	0xfffff124	4c 05 01 00	L...
	0xfffff128	03 00 00 00	....
	0xfffff12c	4c 05 01 00	L...
	0xfffff130	04 00 00 00	....
	0xfffff134	4c 05 01 00	L...
	0xfffff138	05 00 00 00	....
	0xfffff13c	4c 05 01 00	L...
	0xfffff140	06 00 00 00	....
	0xfffff144	b0 05 01 00	....
	0xfffff148	06 00 00 00	....

str lr, [sp, #4] 명령어 수행 -> 현재 sp값 + 4의 위치에 link register 값을 저장했다.

	address	hex	char
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			

  

more	address	hex	char
	0xfffff110	00 00 00 00	....
	0xfffff114	4c 05 01 00	L...
	0xfffff118	01 00 00 00	....
	0xfffff11c	4c 05 01 00	L...
	0xfffff120	02 00 00 00	....
	0xfffff124	4c 05 01 00	L...
	0xfffff128	03 00 00 00	....
	0xfffff12c	4c 05 01 00	L...
	0xfffff130	04 00 00 00	....
	0xfffff134	4c 05 01 00	L...
	0xfffff138	05 00 00 00	....
	0xfffff13c	4c 05 01 00	L...
	0xfffff140	06 00 00 00	....
	0xfffff144	b0 05 01 00	....
	0xfffff148	06 00 00 00	....
	0xfffff14c	00 00 00 00	....

str r0, [sp, #0] 명령어 수행 -> 현재 sp값의 위치에 r0(0)을 저장했다.



1	.text	address	hex	char
2	.global factRecursive			
3				
4	factRecursive:	more		
5	sub sp, sp, #8	0xfffff110	00 00 00 00	....
6	str lr, [sp, #4]	0xfffff114	4c 05 01 00	L...
7	str r0, [sp, #0]	0xfffff118	01 00 00 00	....
8	cmp r0, #1	0xfffff11c	4c 05 01 00	L...
9	bge loop	0xfffff120	02 00 00 00	....
10	mov r0, #1	0xfffff124	4c 05 01 00	L...
11	add sp, sp, #8	0xfffff128	03 00 00 00	....
12	mov pc, lr	0xfffff12c	4c 05 01 00	L...
13	loop:	0xfffff130	04 00 00 00	....
14	sub r0, r0, #1	0xfffff134	4c 05 01 00	L...
15	bl factRecursive	0xfffff138	05 00 00 00	....
16	mov r12, r0	0xfffff13c	4c 05 01 00	L...
17	ldr r0, [sp, #0]	0xfffff140	06 00 00 00	....
18	ldr lr, [sp, #4]	0xfffff144	b0 05 01 00	....
19	add sp, sp, #8	0xfffff148	06 00 00 00	....
20	mul r0, r12, r0	0xfffff14c	00 00 00 00	....
21	mov pc, lr			
22	.end			
23				
24	(end of file)			

**cmp r0, #1과 bge loop 명령어 수행** -> r0와 1의 값을 비교, r0값이 0으로 1보다 작으므로 branch 하지 않고 다음 명령어 mov r0, #1로 이동한다.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24

```
.text
.global factRecursive

factRecursive:
    sub    sp, sp, #8
    str    lr, [sp, #4]
    str    r0, [sp, #0]
    cmp    r0, #1
    bge    loop
    mov    r0, #1
    add    sp, sp, #8
    mov    pc,lr

loop:
    sub    r0, r0, #1
    bl     factRecursive
    mov    r12, r0
    ldr    r0, [sp, #0]
    ldr    lr, [sp, #4]
    add    sp, sp, #8
    mul    r0, r12, r0
    mov    pc, lr

.end

(end of file)
```

signals

registers

name	value (hex)	value (decimal)	descri
r0	0x1	1	
r1	0x0	0	
r2	0x0	0	
r3	0x0	0	
r4	0x10d14	68884	
r5	0x10158	65880	
r6	0x8b408	570376	
r7	0x10158	65880	
r8	0x0	0	registe (64-bit
r9	0x0	0	registe (64-bit
r10	0x89b04	563972	registe (64-bit
r11	0xfffff154	4294898004	registe (64-bit

**mov r0, #1 명령어 수행** -> r0에 1의 값을 넣었다. 이제 r0는 1이다.

1	.text	r6	0x8b408	570376
2	.global factRecursive	r7	0x10158	65880
3		r8	0x0	0
4	factRecursive:	r9	0x0	0
5	sub sp, sp, #8	r10	0x89b04	563972
6	str lr, [sp, #4]	r11	0xfffff154	4294898004
7	str r0, [sp, #0]	r12	0x25	37
8	cmp r0, #1	sp	0xfffff118	4294897944
9	bge loop	lr	0x1054c	66892
10	mov r0, #1	pc	0x10540	66880
11	add sp, sp, #8	cpsr		
12	mov pc, lr	d0		
13	loop:			
14	sub r0, r0, #1			
15	bl factRecursive			
16	mov r12, r0			
17	ldr r0, [sp, #0]			
18	ldr lr, [sp, #4]			
19	add sp, sp, #8			
20	mul r0, r12, r0			
21	mov pc, lr			
22	.end			

add sp, sp, #8 명령어 수행 -> sp에 8을 더한다.(pop동작)

1	.text	r6	0x8b408	570376	
2	.global factRecursive	r7	0x10158	65880	
3		r8	0x0	0	register (64-bit)
4	factRecursive:	r9	0x0	0	register (64-bit)
5	sub sp, sp, #8	r10	0x89b04	563972	register (64-bit)
6	str lr, [sp, #4]	r11	0xfffff154	4294898004	register (64-bit)
7	str r0, [sp, #0]	r12	0x25	37	register (64-bit)
8	cmp r0, #1	sp	0xfffff118	4294897944	
9	bge loop	lr	0x1054c	66892	
10	mov r0, #1	pc	0x1054c	66892	
11	add sp, sp, #8	cpsr			
12	mov pc, lr	d0			VFP double precision (Temp register)
13	loop:				
14	sub r0, r0, #1				
15	bl factRecursive				
16	mov r12, r0				
17	ldr r0, [sp, #0]				
18	ldr lr, [sp, #4]				
19	add sp, sp, #8				
20	mul r0, r12, r0				
21	mov pc, lr				
22	.end				
23					
24	(end of file)				

mov pc, lr 명령어 수행 -> program counter 값을 link register의 값으로 교체한다. link register에는 mov r12, r0의 pc값이 저장되어있다.

show filesystem	fetch disassembly	reload file	jump to line	/home/bbo/CA_2021/lab1/fact_recursive.s:17 (24 lines total)
1	.text			
2	.global factRecursive			
3				
4	factRecursive:			
5	sub sp, sp, #8			
6	str lr, [sp, #4]			
7	str r0, [sp, #0]			
8	cmp r0, #1			
9	bge loop			
10	mov r0, #1			
11	add sp, sp, #8			
12	mov pc, lr			
13	loop:			
14	sub r0, r0, #1			
15	bl factRecursive			
16	mov r12, r0			
17	ldr r0, [sp, #0]			
18	ldr lr, [sp, #4]			
19	add sp, sp, #8			
20	mul r0, r12, r0			
21	mov pc, lr			
22	.end			
23				
24	(end of file)			

name	value (hex)	(decimal)
r0	0x1	1
r1	0x0	0
r2	0x0	0
r3	0x0	0
r4	0x10d14	68884
r5	0x10158	65880
r6	0x8b408	570376
r7	0x10158	65880
r8	0x0	0
r9	0x0	0
r10	0x89b04	563972
r11	0xfffff154	4294898004
r12	0x1	1
sp	0xfffff118	4294897944

mov r12, r0 명령어 수행 -> r12에 r0의 값을 넣는다. r12는 이제 1이다.

1	.text			
2	.global factRecursive			
3				
4	factRecursive:			
5	sub sp, sp, #8			
6	str lr, [sp, #4]			
7	str r0, [sp, #0]			
8	cmp r0, #1			
9	bge loop			
10	mov r0, #1			
11	add sp, sp, #8			
12	mov pc, lr			
13	loop:			
14	sub r0, r0, #1			
15	bl factRecursive			
16	mov r12, r0			
17	ldr r0, [sp, #0]			
18	ldr lr, [sp, #4]			
19	add sp, sp, #8			
20	mul r0, r12, r0			
21	mov pc, lr			
22	.end			
23				
24	(end of file)			

name	value (hex)	(decimal)
r0	0x1	1
r1	0x0	0
r2	0x0	0
r3	0x0	0
r4	0x10d14	68884
r5	0x10158	65880
r6	0x8b408	570376
r7	0x10158	65880
r8	0x0	0
r9	0x0	0
r10	0x89b04	563972
r11	0xfffff154	4294898004
r12	0x1	1
sp	0xfffff118	4294897944

ldr r0, [sp, #0] 명령어 수행 -> sp에 위치한 값을 r0에 저장한다. 현재 0xfffff118에는 1이 저장되어있다. 따라서 r0는 1이 된다.

1	.text	r2	0x0	0	
2	.global factRecursive	r3	0x0	0	
3		r4	0x10d14	68884	
4	factRecursive:	r5	0x10158	65880	
5	sub sp, sp, #8	r6	0x8b408	570376	
6	str lr, [sp, #4]	r7	0x10158	65880	
7	str r0, [sp, #0]	r8	0x0	0	register (64-bit)
8	cmp r0, #1	r9	0x0	0	register (64-bit)
9	bge loop	r10	0x89b04	563972	register (64-bit)
10	mov r0, #1	r11	0xfffff154	4294898004	register (64-bit)
11	add sp, sp, #8	r12	0x1	1	register (64-bit)
12	mov pc, lr	sp	0xfffff118	4294897944	
13	loop:	lr	0x1054c	66892	
14	sub r0, r0, #1	pc	0x10558	66904	
15	bl factRecursive	cpsr			
16	mov r12, r0				
17	ldr r0, [sp, #0]				
18	ldr lr, [sp, #4]				
19	add sp, sp, #8				
20	mul r0, r12, r0				
21	mov pc, lr				
22	.end				
23					
24	(end of file)				

ldr r0, [sp, #4] 명령어 수행 -> sp + 4에 위치한 값을 lr에 저장한다. 현재 0xfffff11c에는 0x1054c가 저장되어있다. 따라서 lr은 0x1054c가 된다.

1	.text	r2	0x0	0	
2	.global factRecursive	r3	0x0	0	
3		r4	0x10d14	68884	
4	factRecursive:	r5	0x10158	65880	
5	sub sp, sp, #8	r6	0x8b408	570376	
6	str lr, [sp, #4]	r7	0x10158	65880	
7	str r0, [sp, #0]	r8	0x0	0	register (64-bit)
8	cmp r0, #1	r9	0x0	0	register (64-bit)
9	bge loop	r10	0x89b04	563972	register (64-bit)
10	mov r0, #1	r11	0xfffff154	4294898004	register (64-bit)
11	add sp, sp, #8	r12	0x1	1	register (64-bit)
12	mov pc, lr	sp	0xfffff120	4294897952	
13	loop:	lr	0x1054c	66892	
14	sub r0, r0, #1	pc	0x1055c	66908	
15	bl factRecursive	cpsr			
16	mov r12, r0				
17	ldr r0, [sp, #0]				
18	ldr lr, [sp, #4]				
19	add sp, sp, #8				
20	mul r0, r12, r0				
21	mov pc, lr				
22	.end				
23					
24	(end of file)				

add sp, sp, #8 명령어 수행 -> sp에 8을 더한다. (pop동작)

	name	value (hex)	value (decimal)	descri
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				

  

	name	value (hex)	value (decimal)	descri
	r0	0x1	1	
	r1	0x0	0	
	r2	0x0	0	
	r3	0x0	0	
	r4	0x10d14	68884	
	r5	0x10158	65880	
	r6	0x8b408	570376	
	r7	0x10158	65880	
	r8	0x0	0	registe (64-bit)
	r9	0x0	0	registe (64-bit)
	r10	0x89b04	563972	registe (64-bit)
	r11	0xfffff154	4294898004	registe (64-bit)
	r12	0x1	1	registe (64-bit)
	sp	0xfffff120	4294897952	

**mul r0, r12, r0 명령어 수행** -> r0에 r0와 r12를 곱한 값을 저장한다. r0가 1, r12가 1이므로 r0에 1이 저장된다.

	name	value (hex)	value (decimal)	descri
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				

  

	name	value (hex)	value (decimal)	descri
	r0	0x1	1	
	r1	0x0	0	
	r2	0x0	0	
	r3	0x0	0	
	r4	0x10d14	68884	
	r5	0x10158	65880	
	r6	0x8b408	570376	
	r7	0x10158	65880	
	r8	0x0	0	registe (64-bit)
	r9	0x0	0	registe (64-bit)
	r10	0x89b04	563972	registe (64-bit)
	r11	0xfffff154	4294898004	registe (64-bit)
	r12	0x1	1	registe (64-bit)
	sp	0xfffff120	4294897952	

**mov pc, lr 명령어 수행** -> program counter 값을 link register의 값으로 교체한다. link register에는 mov r12, r0의 pc값이 저장되어있다. 이를 반복하다보면 r0가 720(r12에 6이 저장된후 계산)되고, lr에 저장된 값에 따라 main의 fact함수로 결과 값을 가지고 복귀한다.

## 2) 실습 B

8, 9번코드에서 **cmp r0, #1, bge loop** 명령어를 수행한다.

bge는 greater or equal, 크거나 같으면 branch 시킨다.

일반적인 경우 **N=V일 때, bge의 조건이 만족되어 branch하게 된다.**

- ① r0가 2 이상의 양수일 때, sub r0, #1에서  $N = 0, V = 0, Z = 0, C = 0$  으로 branch한다.
- ② r0가 1일 때, sub r0, #1에서  $N = 0, V = 0, Z = 1, C = 0$  으로 branch한다.
- ③ r0가 0 이하의 정수일 때, sub r0, #1에서  $N = 1, V = 0, Z = 0, C = 0$  으로 branch하지 않고 다음 명령어로 이동한다.

또한 operand2가 1로 고정되어 있으므로, overflow가 발생할 경우는 하나밖에 없다.

- ① r0가 음수이고 sub r0, #1에서 overflow발생한다.  $N = 0, V = 1, Z = 0, C = 1$  이므로 branch하지 않고 다음 명령어로 이동한다.